RSA Laboratories

Bulletin

number 13 - April 2000

News and Advice from RSA Laboratories

A Cost-Based Security Analysis of Symmetric and Asymmetric Key Lengths

Robert D. Silverman, RSA Laboratories*

Abstract. Recently, there has been considerable debate concerning key sizes for public-key based cryptographic methods. Included in the debate have been considerations about equivalent key sizes for different methods and considerations about the minimum required key size for different methods. In this paper we propose a method of analyzing key sizes based upon the value of the data being protected and the cost of breaking keys.

I. Introduction

A. WHY IS KEY SIZE IMPORTANT?

In order to keep transactions based upon public key cryptography secure, one must ensure that the underlying keys are sufficiently large as to render the best possible attack infeasible. However, this really just begs the question as one is now left with the task of defining 'infeasible'. Does this mean infeasible given access to (say) most of the Internet to do the computations? Does it mean infeasible to a large adversary with a large (but unspecified) budget to buy the hardware for an attack? Does it mean infeasible with what hardware might be obtained in practice by utilizing the Internet? Is it reasonable to assume that if utilizing the entire Internet in a key breaking effort makes a key vulnerable that such an attack might actually be conducted? If a public effort involving a substantial fraction of the Internet breaks a single key, does this mean that similar sized keys are unsafe? Does one need to be concerned about such public efforts or does one only need to be concerned about possible private, surreptitious efforts? After all, if a public attack is known on a particular key, it is easy to change that key. We shall attempt to address these issues within this paper.

Standards, such as ANSI X9.30, X9.31, X9.42, and X9.44 and FIPS 186-2 all require a minimum of a 1024-bit RSA or Diffie-Hellman key. The fundamental question we will answer in this paper is: How long will such keys be secure?

B. WHAT AFFECTS SECURITY REQUIREMENTS?

It is clear that the size of a key must be tied to the value of the data being protected by the key and also tied to the expected lifetime of that data. It makes no sense for an adversary to spend (say) $10 million breaking a key if recovering the key will only net (say) $10 thousand. This principle also applies to keys used to protect other keys, such as the master signature key of a Certifying Authority although such a key is certainly worth more than $10 thousand. Furthermore, if the lifetime of the key or the data being protected is measured in only days or weeks, there is no need to use a key that will take years to break. While standards specify a minimum of 1024 bits for RSA, there are many applications for which 768 bits is more than adequate. While signatures on contracts might need to be secure for 30 years or more [unless time stamped and periodically renewed], other applications can require much less: anywhere from about 1 day for signatures with short-term session keys (such as SSL) to perhaps several years for commercial financial data. Military and national intelligence data such as the identity of spies can have a lifetime of 100 years, but such data is not accessible on-line nor is it protected by public-key methods.

C. BASIC ASSUMPTIONS

Throughout this paper all extrapolations shall be based upon extrapolating speed and memory enhancements of existing hardware and improvements in existing algorithms. While breakthroughs in both algorithm and hardware technology may occur, such events are inherently unpredictable and we do not consider them. Instead, current key sizes are advocated which allow some margin for error.

We also do not believe that any public key size specified today should be used to protect something whose lifetime is more than 20 years. Later in this paper we will advocate a pro-active security policy whereby keys are renewed as the state of the art in breaking keys advances.

II. Methods of Attack

A. GENERAL METHODS FOR RSA AND DL: THE NUMBER FIELD SIEVE

The General Number Field Sieve (GNFS or just NFS) is a general purpose algorithm for either factoring large integers or for solving an ordinary discrete logarithm problem. Its run time depends only on the size of the number being factored, or the size of the underlying field for the discrete logarithm problem. It is the fastest method known today and has two phases. In the first phase, a sieving operation requiring moderately large amounts of memory is conducted on independent computers. This phase is used to construct a set of equations. In the second phase a large Cray with massive memory has then been used to solve this set of equations. Once a solution has been found to this set of equations factoring the number or solving the discrete log takes a miniscule amount of time and memory.

The amount of time it takes to factor a number of x bits is asymptotically the same as the time it takes to solve a discrete log over a field of size x bits. However in practice solving discrete log problems has been more difficult than factoring equivalent numbers. While there are several reasons for this, the main reason is that solving the matrix for a discrete log must be done using multi-precision integer arithmetic while the matrix for factoring is solved mod 2 and thus one can use simple bit operations. It has been estimated that for large x, one can break a discrete log of size x-30 in about the same time as factoring an x-bit number. Throughout this paper we shall assume that solving the two problems are equivalent under NFS and henceforth shall only discuss factoring.

A.1 Complexity of Attacks

The TIME required to factor N using the General Number Field Sieve is

$L(N) = \exp((c + o(1)) ((\log N)^{1/3} (\log \log N)^{2/3}))$.

This function combines the time to do the sieving and the time to solve the matrix. Theoretically each takes an equal amount of time. However, for problems done thus far solving the matrix has taken only a fraction of the time to do the sieving. However, this fraction has been increasing as the numbers get larger.

The SPACE required scales with SQRT(L(N)).

Once you have a benchmark for a particular N, you can predict the difficulty of factoring a larger number N' relative to the difficulty of factoring N by computing L(N')/L(N) to get a time estimate and SQRT(L(N')/L(N)) to get a space estimate. Estimating the ratio this way ignores the effect of the o(1) term in the exponent, but Silverman [6] showed that at least for the Special Number Field Sieve, this term is very small. It does not materially affect the results given herein if its value at 512 bits is (say) .05, while its value at 1024 bits is .01. The results of [6] suggest that these values are not far off.

A.2 Hardware Requirements and Availability

A.2.1 Sieving

The sieving phase of the number field sieve depends for its speed upon the ability to rapidly retrieve values from memory, add to them, and put them back. Therefore, the size of the data cache and speed of the memory bus have a strong impact upon the speed with which the sieve can operate. Furthermore, the sieve operation requires moderately large (but manageable within the state-of-the-art) memory. For a 512-bit key, 64 Mbytes per sieve machine was adequate. However, as the size of the key (and hence the run time) increases, so does the memory required. As shown above, required memory scales with the square root of the run time. Thus, a problem 10 times more difficult than RSA-512 (in terms of time) would require sqrt(10) times more memory [for both phases], or about 200 Mbytes for the sieve machines. If one uses a cost-based model, as we suggest below, the cost of memory very quickly becomes the binding constraint in terms of how much hardware can be acquired.

Historically, as machines became faster, sieving speed did not keep up with machine speed. The main reason for this is that while CPU's were getting faster, memory bus speeds and the sizes of data caches were not keeping pace with speed improvements. However, recent progress in increasing the size of the data cache with each new generation of Pentiums, along with increases in bus speed has alleviated this difficulty. Indeed, reference [1, p. 19] even noted a super-linear increase in speed when moving from a Pentium I to a Pentium II based computer. Lenstra and Verheul suggest that this is due to processor improvements, but we do not believe this viewpoint is correct. The super-linear speed increase can be more readily explained by an increase in cache size and an increase in memory bus speed from 66 MHz to 100 MHz. A fundamental question is therefore: will cache sizes and memory bus speed continue to scale according to Moore's law? There is reason to believe the answer is NO. Lenstra and Verheul's modification of Moore's law says that total CPU cycles doubles every 18 months rather than just doubling processor speed every 18 months. The modification combines an increase in machine speed with an increase in the number of available machines. It seems clear that if one only considers increases in machine speed, without allowing the number of machines to increase one cannot achieve a doubling every 18 months. There is data to support this viewpoint. The VAX, a nominal 1-MIPS machine, introduced in 1977 had a memory bus which ran at 13 MHz. Now the latest generation of PC's has bus speeds of 133 MHz. This is far short of a doubling every 18 months. Further, 10 years ago a state-of-the-art workstation such as a Sparc-10 had an available data cache of 256 Kbytes. Today's Pentium based PC's have data caches

typically around 512 Kbytes. This too falls far short of a doubling every 18 months. Thus, while processor speed increases have followed Moore's law over the last 20 years, other parts of the computer which influence sieving speed have not kept pace. See reference [5] for a more complete discussion of this.

Unless improvements in bus speed and cache size keep pace with improvements in CPU speed, we will once again return to the situation where improvements in processor speed do not help very much in speeding sieving.

Another issue involved in sieving hardware is the size of the required memory. Today's 32-bit machines typically can only address 2 Gbytes of user address space. This represents a factor of only 32 over the memory needed for RSA-512. Data given in section III shows that once keysizes exceed about 710 bits, the memory needed for sieving will no longer be able to be addressed on 32-bit computers. We note that while 64-bit processors are available, it does not appear likely that they will become sufficiently widespread to be useful in attacks for some time to come. There is also a (somewhat speculative) concern about whether the market will routinely demand multi-gigabyte machines. There are few applications today which require multi-gigabyte memories. Servers are one such, but they are not available as distributed sieving machines; they have little idle time and are dedicated to other purposes. While they might contribute some CPU time, the proportion of their processing power so contributed would be limited by their other processing demands. It is certainly questionable whether desktop machines with memories in the 200 Gbyte range will become available anytime soon. As may be seen in section III, 170 Gbytes per machine is what is needed to conduct an attack on 1024-bit keys.

A.2.2 Linear Algebra

The assumptions made in II A, above, imply that the solving the matrix can be done perfectly in parallel. If we can only efficiently solve matrices on small sets of machines running in parallel it means that each of these machines will need not just large, but massive RAM memories. To do a 1024-bit key will require about 6 Terabytes of memory to hold the matrix. We do not know of the existence of a single machine today, or a tightly coupled set of machines with this much memory. Indeed, 6 Tbytes is beyond the ability of 32-bit machines to even address even if distributed among several hundred machines. Trying to estimate when such a machine might become available requires a crystal ball. However, throughout the rest of this paper we assume that the available machines for sieving can somehow be tightly coupled together and used for solving the matrix. This assumption is unrealistic, but allows us to be conservative in discussion of key sizes.

While historically a large Cray has been used to solve the equations, there is no theoretical reason why a set of tightly coupled parallel processors could not do the same thing. Montgomery [5] is experimenting with such a implementation now. Solving large linear algebra problems in parallel has been a topic of research in computer science for a long time. Many papers have been written and there is unanimous agreement that such implementations do not scale well. As one increases the number of processors, communication becomes a bottleneck and total speedup rapidly departs from linear. Indeed, Montgomery's early results do not look encouraging. He reported less than 20% per processor utilization even on a very small (16 machines) tightly coupled fast network. And the drop in per-processor utilization could readily be observed in going from 8 to just 16 processors. While one might theoretically develop a custom-purpose machine for solving such problems, to date no one has proposed such a machine. Its design is still an unsolved research problem.

The Block Lanczos algorithm, which is the algorithm currently employed, is also very close to the theoretically best possible algorithm for solving large matrices mod 2. The best that can be done theoretically, in terms of the number M of rows in the matrix is $O(M^{2+\Sigma})$. The Block Lanczos algorithm already achieved this with $\Sigma \sim .2$ while breaking RSA-512. Therefore the prospects for improved algorithms for solving the matrix do not look very good. One of the assumptions of the Lenstra and Verheul [1] paper is that algorithmic improvements in factoring will continue. It would seem therefore that this assumption of theirs is not correct unless future factoring algorithms can either reduce the size of the matrix, or eliminate the need for solving it. However, the assumption of this paper is that we only extrapolate from existing algorithms.

It is also quite difficult to predict exactly how much faster (say) 1000 machines running in parallel will be than 1 machine. Therefore, throughout this paper we make the unrealistic, but very conservative estimate (conservative with respect to key size) that one can achieve linear speedup when solving the matrix in parallel.

If anything, recommending key sizes based on this assumption will overstate what is possible. Note also, that solving the matrix requires a different parallel architecture than the one assumed for sieving. While sieving each machine runs independently. If a machine or group of machines becomes unavailable, it only means that the sieving slows down by a little bit; the other sieve machines can continue their work. However solving the matrix requires that machines share data frequently. Thus, a single machine stopping can stall all the other machines because they are waiting for data to be forthcoming from the stalled machine. Also, a LAN does not have the bandwidth and its latency is too high to support a parallel group of machines that must communicate frequently. Therefore any parallel matrix solver must be done on a dedicated set of machines with a very fast interconnection scheme. Such a machine must also be fault-tolerant. If a processor goes down, then its current work assignment must be reassigned. Then the other processors must wait while the lost computation is recomputed. It also takes some time to detect a lost processor. Each time it happens a latency is incurred before the entire computation can continue. As the number of machines increases, the probability of a machine being down at any given time increases. A machine being down does not have to be due to hardware problems. In a distributed network (even tightly coupled) sets of machines can become unavailable because of firewall maintenance, software installations/reboots, power failures, someone moving a machine, etc. etc. These problems create strong practical difficulties in getting this kind of application to work. We know of no successful effort to solve linear algebra problems in parallel that involved more than just a few thousand processors at once.

The idea of using 'idle' time on a loosely coupled network simply will not work. The machines need to be tightly coupled, they need to be dedicated to just this task, and they require special interconnection hardware that goes beyond what is provided by (say) a local FDDI net or the Internet. Reference [1] assumes [page 18] that the computers needed for attacks will be available for free. This assumption is reasonable for the sieving phase, but it is totally unrealistic for the linear algebra phase. To do a 1024-bit key will require a tightly coupled parallel computer with 6 Terabytes of memory and a fast interconnection scheme. This kind of hardware costs a lot of money.

B. SPECIAL METHODS FOR RSA AND DL

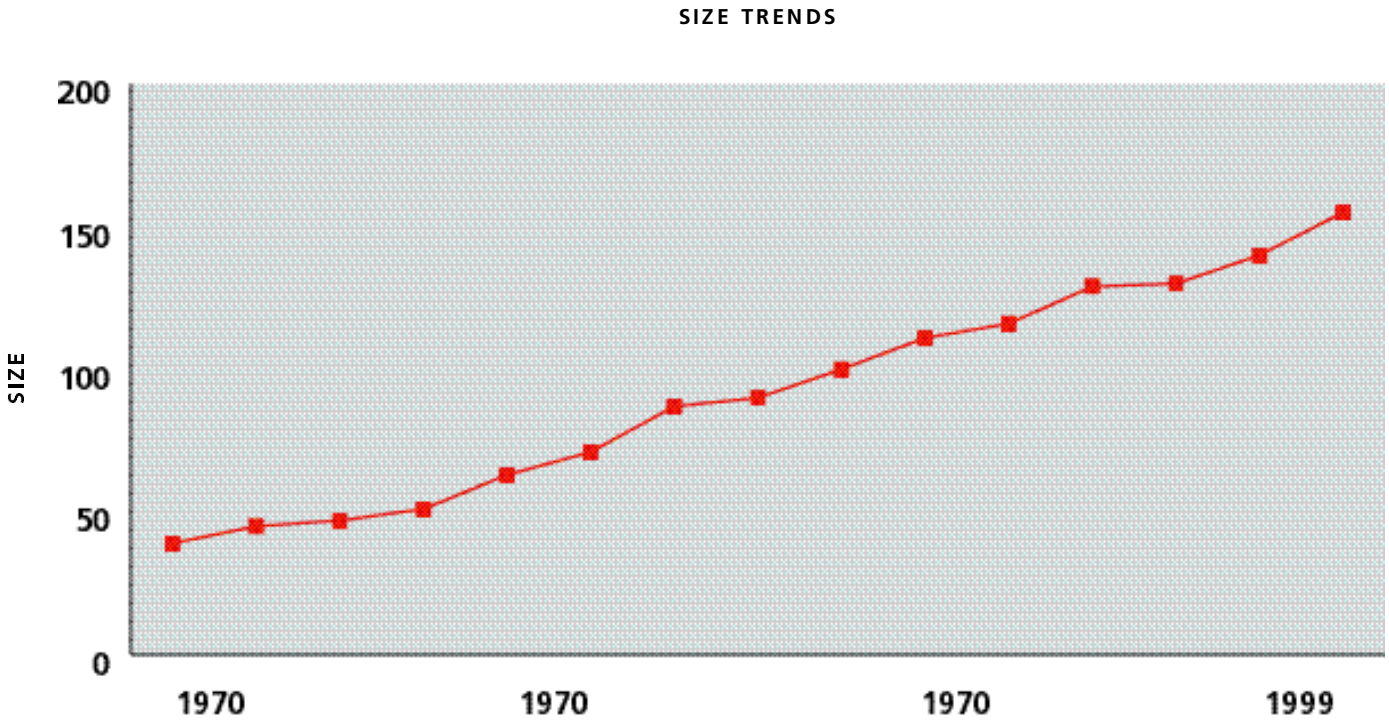
Special methods, such as the Elliptic Curve factoring Method (ECM) have virtually no chance of succeeding in factoring reasonably large RSA keys. For example, the amount of work needed to factor a 1024-bit modulus with ECM is about 10^{17} MIPS-Years and only succeeds with probability $1 - 1/e$. The Number Field Sieve is roughly 10 million times faster. While the ECDL machines of Wiener [section V.] can be trivially modified to run ECM factoring rather than DL, the amount of arithmetic is still unrealistic. The Number Field Sieve, run on ordinary computers is faster. We do not elaborate further on this subject, but refer the interested reader to reference [8].

III. Historical Results and the RSA Challenge

We give below some historical results for record factorizations using a general purpose method. While in many cases the number being factored had special form, the method of attack did not depend on this special form. The data is tabulated and plotted below. Size is given in decimal digits. The Number is sometimes a co-factor of the listed number, after small primes have been divided out.

TABLE 1: Historical Factoring Records

Year	Size	Number	Who	Method	Hardware
1970	39	$2^{128} + 1$	Brillhart/Morrison	CFRAC	IBM Mainframe
1978	45	$2^{223} - 1$	Wunderlich	CFRAC	IBM Mainframe
1981	47	$3^{225} - 1$	Gerver	QS	HP-3000
1982	51	$5^{91} - 1$	Wagstaff	CFRAC	IBM Mainframe
1983	63	$11^{93} + 1$	Davis/Holdridge	QS	Cray
1984	71	$10^{71} - 1$	Davis/Holdridge	QS	Cray
1986	87	$5^{128} + 1$	Silverman	MPQS	LAN Sun-3's
1987	90	$5^{160} + 1$	Silverman	MPQS	LAN Sun-3's
1988	100	$11^{104} + 1$	Internet	MPQS	Distributed
1990	111	$2^{484} + 1$	Lenstra/Manasse	MPQS	Distributed
1991	116	$10^{142} + 1$	Lenstra/Manasse	MPQS	Distributed
1992	129	RSA-129	Atkins	MPQS	Distributed
1996	130	RSA-130	Montgomery	GNFS	Distributed
1998	140	RSA-140	Montgomery	GNFS	Distributed
1999	155	RSA-512	Montgomery	GNFS	Distributed



One thing surprising about this data is that it is VERY linear. A simple least-squares fit yields the equation: Size = 4.23 * (Year – 1970) + 23. The correlation coefficient is about .955. This is somewhat puzzling. The algorithms are sub-exponential. That is, their run time grows more slowly than an exponential function. Moore's law is strictly exponential. It would seem, therefore, based on theoretical grounds that this curve should be growing FASTER than linearly. A possible explanation for this is that even though records continue to be established, that over time the level of effort applied to each effort has dropped. Breaking RSA-129 required a large effort involving many thousands of machines widely distributed over the Internet. Compared with that, breaking RSA-512 was done by a relatively small group of people using far fewer (albeit much faster) machines. If we solve for when a 1024-bit number may be expected to be factored, based solely on extrapolating this historical data, we get an answer of around 2037.

Brent, arguing from theoretical grounds in [3], assumes that given Moore's law, (keysize)1/3 should grow linearly with time. He derives the equation:

Year = 13.25 * $(SIZE)^{1/3}$ + 1928 (SIZE in digits) and extrapolates that 1024-bits should become breakable in 2018. We agree with his theoretical model, but strongly suggest that the historical data shows that key sizes grow linearly with time based upon public efforts. Brent further states that Moore's law suggests 6-7 digits/year advancement in key sizes. We note however, that the historical data suggests instead a growth of about 4.25 digits/year. However, both Brent's estimate and ours suggest that 1024 bits keys should be safe for a minimum of 20 years from public efforts. It is impossible to say what private, unpublicized efforts might have achieved.

As a basis of comparison we shall use data from the break of RSA-512. This effort required a total of 8000 MIPS-Years to do the sieving, represented by about 300 PC's averaging 400 MHz and with at least 64 Mbytes of RAM, running for 2 months, and 10 days and 2.3 Gbytes of memory on a Cray C90 to solve the matrix. Using this data we can predict how much harder it is to factor a number of 576, 640, 704, 768, 1024 or 2048 bits:

Time	Space
$L(2^{576})/L(2^{512}) \sim 10.9$	$SQRT(L(2^{576})/L(2^{512})) \sim 3.3$
$L(2^{640})/ L(2^{512}) \sim 101$	$SQRT(L(2^{640})/L(2^{512})) \sim 10.0$
$L(2^{704})/ L(2^{512}) \sim 835$	$SQRT(L(2^{704})/L(2^{512})) \sim 29$
$L(2^{768})/L(2^{512}) \sim 6 \times 10^{3}$	$SQRT(L(2^{768})/L(2^{512})) \sim 77$
$L(2^{1024})/L(2^{512}) \sim 7 \times 10^{6}$	$SQRT(L(2^{1024})/L(2^{512})) \sim 2650$
$L(2^{2048})/L(2^{512}) \sim 9 \times 10^{15}$	$SQRT(L(2^{2048})/L(2^{512})) \sim 9 \times 10^{7}$

Thus,
576 bits will take 10.9 times as long as RSA-512 and require 3.3 times the memory.
768 bits will take 6100 times as long as RSA-512 and require 77 times the memory.
1024 bits will take 7 million times as long as RSA-512 and require 2650 times the memory.

Note: Space scaling is the same for both the sieving phase and for storing the matrix.

To put this in perspective, it would require about 1.4 billion 500 MHz machines, each with about 170 Gbytes of memory to do the sieving for a 1024-bit number in the same time as RSA-512. While a hacker might try to steal cycles on the Internet by creating a "Number Field Sieve Worm" it is hard to see how such an attack could find enough machines with enough memory to make such an attack feasible. Further, such an attack would be detected and shut down rather quickly as with the Robert Morris worm. Of course increasing speed will reduce the required number accordingly. It would take a single Cray with 6 Terabytes of memory approximately 70 million days (192,000 years)

to solve the matrix. One could reduce this to a mere 19 years with 10000 Crays each with only 600 Mbytes of memory running perfectly in parallel. It is likely that within 10 years common desktop machines will be as fast or faster than a Cray C90 is now. However, it is unlikely in the extreme that 10000 machines running in parallel will be anywhere close to 10000 times as fast as one machine. It would require 10 million such machines running perfectly in parallel to solve the matrix in about the same time as that for RSA-512.

Note that Moore's law assumes that processor speed doubles every 18 months. If one accepts the premise that this will not change, then one expects to see a speed increase of 7 million (needed to do a 1024-bit number relative to RSA-512) in 34 years. This is in fairly close agreement with the estimate of 2037 taken from the historical data above.

It might be argued that this historical data lies BELOW what was theoretically achievable at each data point; that the data represents only modest efforts to break large keys. For example, to do a 600-bit key is about 25 times harder than RSA-512 and would require 5 times the space. Thus, it could be done in the same time as RSA-512 with about 5000 PC's each with 320 Mbytes of memory for the sieving. The Cray C90 would take 250 days and about 11 Gbytes of memory to solve the matrix. Perhaps this is closer to what is theoretically achievable now. We note that 1024-bits is about 292,000 times harder than 600 bits. Based on Moore's law we can expect a factor of 292,000 improvement in about 27 years.

According to PC magazine, approximately 130 Million PC's were sold in 1999. A substantial fraction of these will have been 32 Mbyte machines. We do not have data on the exact fraction. Note that such machines could not even be used for the attack on RSA-512 because they do not have enough memory. It is not unrealistic to assume that with few exceptions machines sold more than 5 years ago simply are not big enough to run an NFS attack. A question we are unable to answer is: how many machines are there on the Internet today that might be both large enough and available for an attack? Lenstra and Odlyzko [9] focus on the total CPU power that might be available, while ignoring that many of those machines are unusable and that among those that are suitable, logistical problems will mean that a fair fraction will not be accessible.

At the rate of 130 million machines/year, it will take more than 10 years for enough machines to be sold to even attempt sieving for a 1024-bit modulus. The number of required machines will decrease as they get faster. However, among these machines most will not have enough memory to be able to be used in an attack. It is also unrealistic to believe that all machines that are sufficiently large will be available for an attack. Other attempts to define key sizes, such as Lenstra's and Odlyzko's [9] have based their estimates on the assumption that most of the CPU cycles on machines attached to the Internet will be available for attacks. These prior estimates have conveniently ignored the fact that most machines simply are not big enough to be used in an attack even if they are available.

IV. Security Estimates for RSA

A. MODELS OF COMPUTATION

A.1 Computational Equivalence

Two algorithms are said to be computationally equivalent if they require the same number of computer operations to complete. Traditional estimates of key size equivalents for different public key algorithms have looked only at computation equivalence. We strongly believe that this approach is wrong because it assumes that CPU TIME is the only constraint that keeps keys from being broken. While this is certainly true if the amount of memory needed to execute an algorithm is negligible, it is not true when memory requirements become prohibitive. For example suppose that key

A1 for algorithm A takes 10 hours to break and the same for key B1 of algorithm B. These keys are equivalent in time. However, if breaking algorithm B requires 10 Gbytes of memory, while algorithm A requires 10 Kbytes, then it is certainly safe to say that algorithm B is harder to break. It requires more hardware and hence costs more. But what if A takes 10 hours and 10 Kbytes, but B takes only 5 hours with 10 Gbytes? Which algorithm is easier to break? The answer would depend on the relative weighting one gave to time versus memory.

A.2 Space Equivalence

When both TIME and SPACE are needed resources to run an algorithm, to equate different algorithms on the basis of TIME only is arbitrary. Why use TIME only? We believe that the answer is twofold. The first is historical – there has always been a historical bias towards time equivalence.

The second reason is ignorance. Many people simply are not aware that complexity theory measures SPACE as well as time, nor are they aware that sometimes an algorithm cannot be run because of space as opposed to time limitations. It is also possible that they assume SPACE problems can be overcome.

Rather than ask: what RSA key size is equivalent to 56-bit symmetric in terms of the time needed to break each key, why not ask: How big does a symmetric key need to be before it is as hard to break as (say) RSA-512 in terms of SPACE? The answer would be millions, if not billions of bits, and this is clearly ridiculous. Yet to measure the problems against one another in terms of SPACE is no more nor less arbitrary than to measure them purely in terms of time. Both time and space are binding constraints and must therefore be considered together.

A.3 Cost Equivalence

While we are in agreement with Lenstra and Verheul that the cost of computer hardware changes with time and is somewhat fluid, we note that the same thing is true of computer speed and memory. It is our belief that measuring key size equivalents in terms of what can be broken in a given amount of elapsed time and with a given amount of money is closer to measuring the true equivalence between different public key methods. We give such estimates in section VIII.

B. DISCUSSION OF MOORE'S LAW

There has been much emphasis placed on Moore's law with respect to predicting what key sizes will be safe 10 or 20 years from now. We note that technology usually follows a sigmoid growth curve; i.e. the curve is S-shaped: there is a ramp-up period in which technology is immature, followed by explosive exponential growth, followed by a gradual tailing off in improvements as technology matures. They question is: where are we with respect to computer technology? Some believe that speed will continue to double every 18 months for the indefinite future, while others such as Stan Williams [Hewlett Packard's chief architect [4]] sees growth slowing down in the near future. The reasons given are that it is becoming progressively harder and more expensive to fabricate smaller circuits and that we are not far off from inherent quantum mechanical limits on MOS-FET silicon technology. While other technologies may hold promise, their viability is hard to predict. For example, there was a time when Josephson junction technology was touted as a cure for silicon limitations, but no one could make it work. Gallium Arsenide has had similar problems.

This paper makes projects based only upon extrapolations of existing technology. It seems safe to assume that Moore's law will continue for 10 years or so. Beyond that seems impossible to predict. More than a 10-fold improvement over existing computers would seem to require new technology and not just incremental improvements on our current techniques. We re-emphasize our conclusion from section III: even if Moore's law continues to hold, it will still require at least 27 years to reach a point where 1024-bit keys are vulnerable to a public effort. The cost of the hardware (many millions of machines even at improved speeds) places a private attack out of reach for all except perhaps governments.

C. PREDICTIONS BASED UPON SHAMIR'S TWINKLE DEVICE

Shamir [2] has proposed a custom purpose device that replaces traditional computers for the sieving operation. This device does seem within current technology to build. The TWINKLE device by itself is however useless. It requires one or more PC's to act as backend processors, feeding it data and retrieving its results. This joint paper with Lenstra reaches the following conclusions:

- TWINKLEs can be built for about $5000 each in quantity.
- To do a 768-bit number requires about 5000 TWINKLEs, supported by 80,000 PCs. The computation would take 6 months.
- They estimate that the 80000 PC's if connected to a fast enough network and if dedicated solely to the task could solve the matrix in 3 months also provided that they were attached to a single, central PC large enough to hold the entire matrix. The central PC would therefore require about 160 Gbytes of memory. They acknowledge that these estimates have not been confirmed by an actual implementation.
- If 768-bits is attempted on ordinary PC's (without TWINKLEs) they write:

"PC's with 5 Gigabyte RAM's which are needed to run the special q siever in standard NFS factorizations are highly specialized. Only a negligible number of such machines exist, and they have few other applications".

From these conclusions, we conclude the following:

- The quoted remark in 4. above is totally inconsistent with assumptions made in reference [1]. Reference [1] deliberately ignores memory problems, saying in effect that memories will grow in size to the point where they will be sufficient for larger keys. But if there are few machines capable of handling 768 bits today, where will machines with 128 Gbyte memories [needed for 1024 bits] come from? Yet Lenstra and Verheul conclude in [1] that 1024 bits will be vulnerable in 2002 using "for free" machines distributed on the Internet. However, this conclusion was reached based on a primary assumption of that paper: that DES was breakable in 1982. We simply observe that DES was not actually broken until 1997. Further, the remark in 4. about having few applications might be taken to imply the following: While it might become theoretically possible to build multi-gigabyte machines at reasonable cost in 10 years, there are few applications which demand such machines. But machine capabilities are driven by market needs. If there is little need for such machines, will they become readily available in 10 years? We do not pretend to have an answer to this question. We also note that one cannot even put 5 Gbytes of RAM on today's 32-bit PC's.
- It does not seem possible to tightly couple 80,000 processors with today's technology. The largest such machines today of which we are aware typically consist of 1000 to 2000 processors at most and are very expensive.
- Doing 1024 bits with TWINKLEs would be 1400 times harder (7 million/5000) than 768 bits. Thus, attempting 1024 bits even with the aid of TWINKLEs still seems way beyond reach.
- If one could do the sieving INFINITELY fast and at zero cost, solving the matrix for 768 bits and beyond is still prohibitive.

V. Elliptic Curve Cryptosystems

A. METHOD OF ATTACK

The best known attack against an Elliptic Curve Discrete Log system is based upon a collision attack and the birthday paradox. One expects that after computing approximately sqrt(order of the curve) points, that one can find two points that are equivalent under an algebraic relationship. From this collision, the key can be found. Thus, the best known attack is purely exponential in the size of the key. The time complexity is:

$T(k) = \text{sqrt}(\text{pi}/2) * 2^{k/2}$, where k is the bitsize of the order of the basepoint.

The space requirements are modest, even for large k.

B. TIME AND COST OF ATTACK

Wiener, in 1996, proposed a special purpose hardware design that for $10 million could break a 155-bit Elliptic Curve key over a 120-bit sub-field in 32 days. The time to do a k-bit Elliptic Curve is then $32 * \text{sqrt}(2^{k-120})$ days with one of these machines. It is likely that a faster machine could be designed and implemented today, and we will assume a machine that is about 50 times faster and can therefore break the given key in about 12 hours rather than 32 days.

VI. Symmetric Key Systems (Private Key Systems)

A. METHOD OF ATTACK

The best known attack against symmetric key systems is a brute-force search of the key space. Nothing else seems to work. Thus, the attack is purely exponential. Thus, for a k-bit symmetric cipher, the expected time to break it is

$T(k) = 2^{k-1}$.

The space requirements are trivial: a few kilobytes suffices.

B. TIME AND COST OF ATTACK

Wiener, in 1993, designed a $1 million DES cracking machine which would crack a DES key in 3.5 hours. This is slow by current technology standards and can easily be improved. We shall assume a machine that is 100 times faster than this one.

VII. RSA Key Size Recommendations

A. NEAR TERM

Breaking a 1024-bit key with NFS is impossible today. Enough sufficiently large machines simply do not exist to do the sieving, and solving the matrix will require a major technology breakthrough. This situation should remain for at least 20 years. No foreseeable increase in machine speed or availability will allow enough hardware to be gathered.

Further, 768 bits seems unreachable today, even utilizing the TWINKLE device, because of the difficulties in dealing with the matrix. However, we do believe that 768-bits might be breakable in about 10-years time. The fitted data from section III gives a date of 2019 for breaking such a key by public effort.

With respect to yet even larger key sizes we note that for a 2048-bit key, the matrix will closely approach the address limits of even a 64-bit processor (10^{18} bytes or so) and that the total data collected during sieving will exceed the address limits. There is no predictable time horizon for when 128-bit computers may appear.

B. PRO-ACTIVE SECURITY

We strongly advocate a policy of pro-active security. Software which uses public keys should not statically define key sizes according to what is infeasible today. Instead, software should provide the capability to instantiate new keys and new key sizes as the art in breaking keys progresses. Systems need to be flexible – to change keys as needed, to resign as needed, to re-protect data as needed and to timestamp where appropriate.

While changing keys and resigning documents protects signatures against cryptanalytic advances, there is no way to protect old data that was publicly transmitted under old keys. Therefore key sizes must be selected carefully according to the lifetime of the data. It will do an adversary no good to break a current key in (say) 10 years, if the data becomes useless after 5 years.

VIII. Cost-Based Key Size Equivalencies

We assume that a PIII processor at 500 MHz can be acquired for $100.00 and that memory costs $.50/Mbyte. These assumptions are slightly optimistic, given current costs but making this choice yields conservative key size estimates. This section presents key size equivalents for RSA, Elliptic Curves, and Symmetric Key systems using a cost-based model. We assume that $10 million is available to conduct an attack.

Consider using Wiener's Elliptic Curve cracker for a 120-bit subfield as a data point in constructing the table below. If one extrapolates downward to 112 bits, this problem is sqrt(2^8) or 16 times easier. It seems that such a machine could break a 112-bit EC key in about 45 minutes. Note that this time estimate is quite sensitive to estimates of the speed of the machine and one has never been built.

We expect that today a machine could be built that is 100 times faster than Wiener's DES machine. Thus, we assume that today one could build a machine for $10 million which would break a DES key in .03 hours or about 100 seconds.

Based upon a purely computational model, the amount of arithmetic needed to break a 56-bit DES key is about the same as that needed to break an EC key which is twice that size: 112 bits. However, the Wiener designed 56-bit DES cracker seems faster than his equivalent 112-bit EC cracker. We take as a base point in the table below the assumption that 56-bit DES could be broken in "about" 5 minutes with the right hardware and that this is indeed equivalent to 112-bit EC.

While the TWINKLE device of Shamir seems to be a very effective way of doing the sieving for RSA keys in the (say) 512 to 700 bit range, even Shamir admits that the device is unlikely to scale to where it is effective in attacking 1024 bit keys. Thus, for the table below we assume a software only attack using PC's for sieving and tightly coupled PC's for the linear algebra. We assume 500-MIPS machines and that the number of such machines available for $10 million is:

$10^7/(100 + .5 * \text{required memory in Mbytes})$

The denominator represent a per machine cost of $100 for the processor plus the cost of the memory. The required memory is assumed to be

64 Mbytes * SQRT($L(2^{keysize})/L(2^{512})$) since 64 Mbytes was required for RSA-512.

We assume that the total memory for all the sieve machines is adequate to hold the matrix and that these same machines can be tightly coupled. Therefore, if we have F dollars to spend on hardware, and time T (in months) for an attack we obtain the following formula:

$$L(N)/L(2^{512}) = \frac{T/2 * F}{300 * (100 + .5\ \text{sqrt}(L(N)/L(2^{512})) * 64)}$$

This formula takes as a baseline that RSA-512 took 2 months on 300 PC's, each with 64 Mbytes of memory. This explains the term T/2 in the numerator and the numbers 300 and 64 in the denominator. The value of N that satisfies the above equation is the modulus that can be broken for $F and time T.

In reality there would be a large additional cost for the fast interconnection network needed for the tightly coupled parallel machine used to solve the matrix, but we ignore this component. Doing this can only make our recommended key sizes more conservative because if we include this cost it means fewer machines are available for $10 million and hence the key size that can be attacked would be smaller.

This table gives key size equivalents assuming that $10 million is available for computer hardware. It assumes that EC key sizes should be twice the Symmetric Key sizes.

TABLE 2: Cost Equivalent Key Sizes

Symmetric Key	EC Key	RSA Key	Time to Break	Machines	Memory
56	112	430	less than 5 minutes	10^5	trivial
80	160	760	600 months	4300	4 Gb
96	192	1020	3 million years	114	170 Gb
128	256	1620(1)	10^{16} yrs	.16	120 Tb

The table above gives cost-equivalent key sizes. It gives the size, in bits, for equivalent keys. The time to break is computed assuming that Wiener's machine can break a 56-bit DES key in 100 seconds, then scaling accordingly. The "Machines" column shows how many NFS sieve machines can be purchased for $10 million under the assumption that their memories cost $.50/Mbyte.

Note: (1) The memory needed for an RSA equivalent to 128-bit symmetric is about 120 Terabytes. Such a machine cannot be built for $10 million, therefore there is no RSA equivalent for $10 million because one cannot build .16 of a machine. Note further that the universe is only 15×10^9 years old. Suppose therefore that instead of allowing $10 million for an attack, we allow $10 trillion. Now, the attack on the symmetric and EC keys takes 'only' 10^{10} years – about two-thirds the lifetime of the universe. The RSA equivalent for this is about 1620 bits. This key is 4×10^{12} times harder than RSA-512. Each machine requires 1.2×10^{14} bytes of memory, and we can purchase about 158000 of them for $10 trillion.

Note that traditionally a strict equivalence has been made between 80-bit symmetric, 160-bit EC and 1024-bit RSA. This equivalence has been based PURELY on the assumption that the only required resource is CPU time. It has ignored the size and the cost of memory needed to break a 1024-bit RSA-key. The large difference in RSA key size (760 bits vs. 1024) comes solely from the fact that for fixed FINANCIAL resources, the cost of memory is by far the largest cost associated with breaking an RSA key and computational equivalence ignores memory.

Here are examples of how the RSA equivalents were computed:

With 100000 machines and 5 minutes instead of 2 months, we can solve an RSA problem that is about 26 times easier than RSA-512. $L(2^{512})/L(2^{430})$ is about 26 if we ignore the time to solve the matrix. Thus, this estimate is conservative.

760 bits is about 4800 times harder than 512 bits. This requires about 4.4 Gbytes per machine. Each one costs $2300, giving about 4300 machines for $10 million. RSA-512 took 2 months with 300 machines, thus 760 bits should take 9600 months with 300 machines or about 670 months with 4300 machines. Thus, the estimate of 760 bits is slightly conservative.

Note:

Changing the amount of money available for an attack does not change key size equivalents. All this does is reduce the time needed for an attack to succeed. Thus, for $100 million breaking a 760 bit RSA key takes 60 months instead of 600 months. But this is still equivalent to an 80 bit symmetric key because the time to break the latter is similarly reduced.

IX. Conclusion

While the Lenstra and Verheul paper [1] reaches the conclusion that 1024-bit RSA keys will be safe only until 2002, we find this conclusion baffling. This conclusion was reached by assuming that 56-bit DES was vulnerable in 1982 despite the fact that it was not until 1997 that DES was actually broken. Does anyone seriously believe that we can solve a problem 7 million times harder than RSA-512 (and needing 6 Terabytes of memory) within just the next few years when RSA-512 was only recently done?

The cost of memory and the difficulty of scaling hardware for solving the matrix suggests that 1024 bit keys will be safe for at least 20 years (barring an unexpected new factoring algorithm). The hardware does not exist today which will allow an NFS attack on a 1024-bit key. Discussion of "total cycles available on the Internet" is irrelevant if machines are not large enough to run NFS.

There are basically four reasons why someone might want to break a cryptographic key:

- Economic gain. For hackers with such a motive, the cost of conducting the attack on suggested key sizes is prohibitive. Such an attack must be conducted in secret, otherwise the supposed victim could just change the key. Hence, the model of doing sieving "for free", on the Internet does not apply because such an attack could not be secret.
- Malice. An attacker might want to break a key to be malicious, but once again such an attack must be done in secret. Is it conceivable that a corporation with large resources might want to break some key from malice or that it could be done in secret? A typical Internet hacker could not possibly obtain the required resources in private.
- Research. An attack might be conducted only to establish a limit on what can be done. But such an attack would not be done on a particular user's key and therefore does not threaten existing keys.
- National Security or Law Enforcement. Citizens should not fear a government attack on a key for economic motives as the cost of attack will exceed the economic gain that might be derived from breaking a key. Citizens do need to be concerned about government intrusions on privacy. It seems to be an unanswerable question as to what level of effort might be expended by a government to retrieve a key for such a purpose.

We suggest that users maintain a flexible, pro-active policy in which they are able to change to larger keys as the art in breaking keys improves.

If one accepts a purely computational model of equivalence, as opposed to a financial model equivalence, then we agree with the key size equivalents given in reference [1]. However, we do not agree with the conclusions about when large RSA keys will be vulnerable.

References

(1) Lenstra, A., and Verheul, E. Selecting Cryptographic Keysizes.
(2) Lenstra, A., and Shamir, A., Analysis and Optimization of the TWINKLE Factoring Device.
(3) Brent, R., Some Parallel Algorithms for Integer Factorization.
(4) Q & A with Stan Williams, Technology Review, Sept 1999 pp. 92-96.
(5) Silverman, R., Exposing the Mythical MIPS-Year, IEEE Computer, Aug 1999 pp. 22-26.
(6) Montgomery, P. Parallel Implementation of the Block-Lanczos Method, RSA-2000 Cryptographers Track.
(7) Silverman, R. The o(1) Term in the Complexity of the SNFS. Rump Session talk, Crypto '97.
(8) Silverman, R., & Wagstaff Jr., S. A Practical Analysis of the Elliptic Curve Factoring Method, Mathematics of Computation, vol. 61, 1993, pages [445-463].
(9) Odlyzko, A., The Future of Integer Factorization, CryptoBytes 1 (1995) pp. 5-12.

* *Robert Silverman is a senior research scientist at RSA Laboratories in Bedford, MA. He has an A.B. from Harvard in Applied Mathematics and a Masters (an ABD) from the University of Chicago in Operations Research. he spent four years at Data Resources Inc. and ten years at the MITRE Corporation where he was a Principal Scientist. His research interests include parallel and massively distributed computing, computational number theory, algorithmic complexity theory and general design and analysis of numerical algorithms. He is a member of the American Mathematical Society.*

X. Appendix: Analysis of Multi-Prime RSA Security

I. INTRODUCTION

Classically, an RSA modulus has been composed from two primes. However, there are very practical reasons why using more than two primes might be preferred.

- The primes are smaller and key generation takes less time despite there being more of them.
- Private key operations take less time if one uses the Chinese Remainder Theorem. Using three primes vs. two primes gives a theoretical speedup of 9/4. A speedup of 1.8 to 2.0 has been achieved in practice.

There are two possible methods for attacking an RSA key if it is built from more than two primes. The first is NFS, which has already been discussed. The second method is the Elliptic Curve Method (ECM). This discussion assumes minimal familiarity with the Elliptic Curve factoring algorithm. A brief, simplified description of the method follows:

One chooses two integers A and B at random and considers the equation $y^2 = x^3 + Ax + B$. This is an elliptic curve. Suppose N is the RSA modulus and that p is a prime dividing N. We need the following definition:

m-Smooth Number – An m- smooth number is one which has all of its prime factors less than m.

The number of points on the randomly chosen elliptic curve taken modulo p is a random integer near p. ECM succeeds when this number of points is m-smooth for a suitably chosen small value of m.

Whereas the time for the Number Field Sieve to factor a number depends *only on the size of the number*, the Elliptic Curve Method finds small prime factors of a larger number and its run time depends *on the size of the factors*.

Throughout this paper it is assumed that an RSA modulus built from more than two primes will (try to) be broken with the Elliptic Curve Method. Therefore the MIPS-Year estimates in sections V. and higher are based upon using ECM. The difficulty of breaking a modulus with ECM is then computed relative to a base point. That base point is the factorization of RSA-512 with NFS.

Definition of MIPS-Year

We define a MIPS-Year to be 3.1×10^{13} arithmetic operations. This is 1×10^{6} operations/sec x 3600sec/hr x 24hrs/day x 365 days/yr x 1 yr.

II. DIFFICULTY IN MIPS-YEARS OF BREAKING RSA WITH THE NUMBER FIELD SIEVE

We have a benchmark data point using NFS for breaking RSA-512 of

8000 MIPS-Years for the sieving using ~50 Mbytes/machine
10 Days on a Cray using 2.3Gbytes to do the matrix.

To measure the difficulty for NFS in MIPS-Years of factoring moduli greater than 512 bits we use:

$8000 * L(N)/L(2^{512})$

where L(N) was defined in the main body of this paper.

III. METHOD OF ANALYSIS FOR ECM

ECM succeeds when a randomly chosen curve (taken over Z/pZ where p | N is the factor we hope to find) has its group order smooth up to B1 for suitably chosen B1. This means that all the prime factors of the order of the curve must be less than B1. As it is usually implemented ECM is run in two steps. In Step 1, one hopes that the order of the curve is smooth up to B1. In Step 2, one hopes that the order is smooth up to B1 except for a single prime factor lying between B1 and B2 for a suitably chosen value of B2. The second step has the effect of about an order-of-magnitude speedup in practice, although it does not affect the asymptotic complexity of the method. Step 1 requires approximately B1 elliptic curve point additions. The number of operations for Step 2 depends upon the way it is implemented – there are several ways of doing so. If one uses Fast Fourier Transforms, Step 2 takes about the same time as Step 1, when $B2 = B1^2$.

The probability that an integer y has all of its prime factors less than x is about u^{-u} where $u = (\log y)/(\log x)$. This is how the probability of success, per curve, is computed herein.

Then, the total work in MIPS-Years to find a factor of an n bit number is .033 * (n/1024)^2 * $B1/10^7$ * u^u. (See below for an explanation of the numbers .033 and 107.) The expression (n/1024)^2 is the relative work factor to execute the multi-precision arithmetic for an n bit number as compared with the 1024 bit benchmark given below. The expression $B1/10^7$ represents the relative work factor to take the computations to B1 relative to the benchmark for $B1 = 10^7$ given below. And u^u is the number of curves needed.

It is assumed that when running (say) k curves, one can give one curve to each of k machines, or two curves to each of k/2 machines etc.

We assume that the modulus has been broken when ECM finds a single factor. Clearly if more than two primes are used this means we still have a composite cofactor. However, pulling out the next factor with ECM is then easier because the modulus is smaller. Thus, one might want to apply ECM again once the first factor is removed, or use NFS if it would then be faster.

IV. ELLIPTIC CURVE BENCHMARK DATA

We have an established benchmark of:

Executing Step 1 to 1.33×10^7 takes 454 seconds on a 500MHz Dec Alpha 21264 for a 127 decimal digit modulus. This translates to 2000 seconds for Step 1 to 10^7 on a 1024 bit modulus. 2000 seconds on this machine translates roughly to about .033 MIPS-Years for this problem. We designate the Step 1 limit by the variable B1 throughout this paper.

Note that this is a 64 bit machine. The same computation on a 32 bit machine, assuming all other architectural features to be the same will take four times as long because the complexity of multi-precision arithmetic decreases with the square of the word size.

This can be sped by dedicated hardware for multi-precision arithmetic. Indeed, the Wiener ECDL cracking machine, discussed in the main body of this paper, is easily adapted to run ECM. We therefore borrow his design for this purpose. This machine has 2.5×10^7 processors and it will compute the addition of two points on an elliptic curve modulo a 155 bit number at the rate of 20,000 additions/sec. Therefore, the time on this machine needed to execute Step 1 of ECM to the limit of B1 is

$B1/20000 * (\text{Size}(N)/155)^2$ where N is the size in bits of the RSA modulus. The term $(N/155)^2$ represents the relative difficulty of doing the arithmetic modulo N, as opposed to the 155 bit arithmetic of the Wiener machine.

While ECM has a second stage that can be implemented, this second stage does not affect the asymptotic run time of the algorithm. It does however, have some practical importance for the size of the numbers under consideration here. Using Fast Fourier Transform techniques can allow computing Step 2 to $B1^2$ in about the same time as executing Step 1 to B1. This can give about an order of magnitude performance increase at twice the time cost. The space requirements, while large, are manageable (128 to 256 Mbytes for 1024 bit moduli per processor). This paper computes ECM costs and probability of success based upon a Step 1 implementation only.

Let p be the probability of success with a single curve. Then the expected number of curves needed to succeed is $1/p$. The probability of succeeding with $1/p$ curves is then $1 - (1 - p)^{(1/p)}$ and this is about $1/e \sim .63$ when p is moderately large. Halving the number of curves yields a probability of success of about .39, while doubling the number of curves yields a probability of success of about .88. Except where noted otherwise, this paper will assume that $1/p$ curves are used. Thus, this method **does not succeed with certainty**.

Note that as B1 increases, the probability of success per curve increases and the required number of curves decreases. But the time required to run each curve increases. There is an optimal choice of B1 depending on the size of the factor being sought. The tables below show where this approximately occurs.

V. ANALYSIS OF 768 BIT RSA MODULUS WITH THREE 256 BIT PRIMES.

B1	Time/curve (hrs)	Probability Per curve	No. of Curves	Total MIPS-Years
10^7	.31	3.4e-12	2.9e11	5.3e9
10^8	3.1	3.3e-10	3.0e9	5.4e8
10^9	31	1.0e-8	9.7e7	1.7e8
10^{10}	310	1.5e-7	6.8e6	1.2e8
10^{11}	3100	1.2e-6	8.4e5	1.5e8

The minimum occurs with B1 $\sim 10^{10}$. With the benchmark DEC Alpha each curve requires 310 hours to compute and has a probability of success of 1.5×10^{-7}. Thus, 6.8 million curves are needed, giving a total work effort of 1.2×10^8 MIPS-Years. One can run each curve for only 31 hours, but then the total number of curves goes up by a factor of 14 and the total work by a factor of about 1.4.

This is 15000 times harder than breaking RSA-512 with NFS. With Step 2 implemented, this might be 3000 to 5000 times harder.

Note that breaking this modulus with NFS is 6100 times harder than RSA-512, so a 768 bit modulus with three primes is as hard to break as one with two primes with respect to **time equivalence only**.

We now do a cost-based analysis based on the Wiener machine. It can execute the algorithm to 10^{10} on a 768 bit modulus in $10^{10}/20000 * (768/155)^2$ seconds which is about 3400 hours (5 months). This machine can try 25 million curves at once during this time but only 6.8 million are needed to succeed with probability .63. If we use 25 million curves, the probability of success is about .974.

Note that running only 6.8 million curves (giving one per processor) leaves many processors idle. One way to rectify this is to run more curves than is optimal (25 million), while slightly lowering the Step 1 limit. This results in more total arithmetic being performed, but will slightly lower the elapsed time. An alternate way to utilize addition processors would be to use them to speed the multi-precision arithmetic on a per-curve basis. However this would require a major redesign of the machine. Similar comments apply to other key sizes as long as the number of needed curves is less than the number of processors on the machine.

VI.A. 1024 BIT RSA MODULUS WITH THREE PRIMES (341 BITS)

B1	Time/curve (hrs)	Probability Per curve	No. of Curves	Total MIPS-Years
10^7	.56	7.9e-18	1.3e17	4.0e15
10^8	5.6	6.0e-15	1.7e14	5.3e13
10^9	56	8.8e-13	1.1e12	3.6e12
10^{10}	560	4.2e-11	2.4e10	7.7e11
10^{11}	5.6e3	8.9e-10	1.1e9	3.6e11
10^{12}	5.6e4	1.1e-8	9.4e7	3.0e11
10^{13}	5.6e5	8.2e-8	1.2e7	3.9e11

The minimum is at B1 ~ 10^{12}. Each DEC Alpha machine requires 56000 hours per curve, 9.4 million curves are required and the total work is 3.0e11 MIPS-Years. This is slightly harder than factoring the same modulus with NFS **based on time-equivalence only**.

If one uses the Wiener machine, it will take about 610,000 hours (70 years) to succeed with probability .86, using 24 million curves to a Step 1 limit of 10^{12}.

We do not show the computations for trying to break a two-prime 1024 bit modulus with ECM. However we note here that it is approximately 10^7 times harder than breaking the same modulus with NFS. Thus, it is clearly not worth attempting.

VI.C. 1024 BIT MODULUS WITH FOUR PRIMES (256 BITS)

This is $(1024/768)^2$ ~ 1.77 times as hard as 768 bits with three primes, i.e. about 9 months to succeed with probability .63.

VII.A. 1536 BIT RSA MODULUS WITH THREE PRIMES (512 BITS)

This is 9/4 as hard as breaking a two-prime 1024 bit modulus with ECM. This is harder than breaking 1536 bits with NFS. The Wiener machine would require about 300 million years to run 12 billion elliptic curves, each to a Step 1 limit of about 10^{16}.

VII.B. 1536 BIT RSA MODULUS WITH FOUR PRIMES (384 BITS)

B1	Time/curve (hrs)	Probability Per curve	No. of Curves	Total MIPS-Years
10^8	12.5	1.7e-17	5.7e16	4.1e16
10^9	125	5.7e-15	1.7e14	1.3e15
10^{10}	1250	5.1e-13	1.9e12	1.4e14
10^{11}	1.2 x 104	1.8e-11	5.4e10	3.9e13
10^{12}	1.2 x 105	3.3e-10	3.0e9	2.2e13
10^{13}	1.2 x 106	3.6e-9	2.7e8	1.9e13
10^{14}	1.2 x 107	2.7e-8	3.7e7	2.7e13

The minimum occurs at B1 ~ 10^{13}. With the DEC Alpha, each curve requires 2.2 Million hours to compute, 270 Million curves are required and the total work is 1.9 x 10^{13}.

This is about 2.4 billion times harder than RSA-512 with NFS, but is easier than breaking 1536 bits with NFS. The Wiener machine would take 60 million hours (7000 years) to run 270 million curves to a Step 1 limit of 10^{13}.

VIII. 2048 BIT RSA MODULUS

We do not show the computation for three primes. The work required is about 10^{22} MIPS-Years.

4 Primes (512 bits) – This is four times harder than RSA-1024 with two primes.

5 Primes (409 bits)

B1	Time/curve (hrs)	Probability Per curve	No. of Curves	Total MIPS-Years
10^8	22	5.3e-19	1.9e18	2.4e18
10^9	220	2.9e-16	3.4e15	4.2e16
10^{10}	2200	3.7e-14	2.6e13	3.3e15
10^{11}	2.2×10^4	1.8e-12	5.5e11	6.9e14
10^{12}	2.2×10^5	4.2e-11	2.4e10	3.0e14
10^{13}	2.2×10^6	5.7e-10	1.8e9	2.3e14
10^{14}	2.2×10^7	5.0e-9	2.0e8	2.6e14

The minimum occurs at B1 ~ 10^{13}. With the DEC Alpha, each curve requires four million hours to compute and 1.8 billion curves are required. The total work is 2.3×10^{14}.

This is about 10^5 times easier than breaking 2048 bits with NFS. The Wiener machine would take 432 million hours (50,000 years) to run 1.8 billion curves to a Step 1 limit of 10^{13}.

IX. CONCLUSIONS

The following table displays the relative difficulties with respect to time only for breaking different keys. It gives the time to break the key in MIPS-Years.

Table A1. Work to break Multi-Prime RSA (MIPS-Years)

Size of Modulus	NFS time to break	3 Primes with ECM	4 Primes with ECM	5 Primes with ECM
768	4.8×10^7	1.2×10^8	too easy	much too easy
1024	5.6×10^{10}	3.0×10^{11}	2.1×10^8	too easy
1536	6.0×10^{15}	4.0×10^{17}	1.9×10^{13}	4.2×10^{10}
2048	7.0×10^{19}	~ 10^{22}	9.0×10^{17}	2.3×10^{14}

In the following table we give approximate symmetric key equivalents for multi-prime RSA assuming that the Wiener machine is available and costs $10 million. If we assume that the benchmark DEC Alpha could be stripped of all components except the mother board and would cost $250 each, we could purchase 40,000 of them. This set of machines would be about 20 times slower than the Wiener machine, but would be readily available.

Table A2. Cost Equivalent Key Sizes with Multiple Primes

Symmetric Key Size (bits)	RSA Modulus Size	Number of Primes	Time to Break
73	768	3	5 months
80	1024	3	70 years
74	1024	4	9 months
103	1536	3	300 million years
90	1536	4	75000 years

Note that this complements Table 2 of the main body of the paper. Here, we see three-prime 1024 bit RSA is about the same complexity as an 80 bit symmetric key with respect to both time and cost. Two-prime 1024 bit RSA is equivalent to an 80 bit symmetric key in terms of time, but much stronger in terms of cost.

Corporate Headquarters

European Headquarters

Asia Pacific Headquarters

April 2000 Bulletin #13